(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 February 2003 (20.02.2003)

PCT

(10) International Publication Number
WO 03/014921 A1

(51) International Patent Classification[7]: G06F 9/44

(21) International Application Number: PCT/US02/20992

(22) International Filing Date: 2 July 2002 (02.07.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/302,891	2 July 2001 (02.07.2001)	US
60/306,376	17 July 2001 (17.07.2001)	US
10/187,858	27 June 2002 (27.06.2002)	US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:

US	60/302,891 (CIP)
Filed on	2 July 2001 (02.07.2001)

(71) Applicants *(for all designated States except US)*: NAZOMI COMMUNICATIONS, INC. [US/US]; 2200 Laurelwood Road, Santa Clara, CA 95054 (US). PATEL, Mukesh, K. [US/US]; 787 Boar Circle, Fremont, CA 94539 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: HILLMAN, Dan [US/US]; 6607 Winterset Way, San Jose, CA 95120 (US). KAMDAR, Jay [US/US]; 10080 Carmen Road, Cupertino, CA 95014 (US). SHIELL, Jon [US/US]; 801 Seabury Drive, San Jose, CA 95136 (US). RAVAL, Udaykumar, R. [IN/US]; 2200 Monroe Street, #1608, Santa Clara, CA 95050 (US).

(74) Agent: O'MALLEY, Joseph, P.; Burns, Doane, Swecker & Mathis, LLP, P.O BOX 1404, Alexandria, VA 22313 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 03/014921 A1

(54) Title: INTERMEDIATE LANGUAGE ACCELERATOR CHIP

(57) Abstract: An accelerator chip can be positioned between a processor chip and a memory (26). The accelerator chip enhances the operation of a Java program by running portions (22) of the Java program for the processor chip (24). In a preferred embodiment, the accelerator chip includes a hardware translator unit and a dedicated execution engine.

[006] Poor Java software performance, particularly in embedded system designs, is a well-known issue and several techniques have been introduced to increase performance. However these techniques introduce other undesirable side effects. The most common techniques include increasing system and/or microprocessor clock frequency, modifying a JVM to compile Java bytecodes and using a dedicated Java microprocessor.

[007] Increasing a microprocessor's clock frequency results in overall improved system performance gains, including performance gains in executing Java software. However, frequency increases do not result in one-for-one increases in Java software performance. Frequency increases also raise power consumption and overall system costs. In other words, clocking a microprocessor at a higher frequency is an inefficient method of accelerating Java software performance.

[008] Compilation techniques (e.g., just in time "JIT" compilation) contribute to erratic performance because the speed of software execution is delayed during compilation. Compilation also increases system memory usage because compiling and storing a Java program consumes an additional five to ten times the amount of memory over what is required to store the original Java program.

[009] Dedicated Java microprocessors use Java bytecode instructions as their native language, and while they execute Java software with better performance than typical commercial microprocessors they impose several significant design constraints. Using a dedicated Java microprocessor requires the system design to revolve around it and forces the utilization of specific development tools usually only available from the Java microprocessor vendor. Furthermore, all operating system software and device drivers must be custom developed from scratch because commercial software of this nature does not exist.

[0010] It is desired to have an embedded system with improved Java software performance.

Summary of the Present Invention

[0011] One embodiment of the present invention comprises a system including at least one memory, a processor chip operably connected to the one memory, and an Accelerator Chip. The memory access for the processor chip to at least one memory being sent through the

to execute native instructions provided by the hardware translator unit. The dedicated execution engine only executing instructions provided by the hardware translator unit. The hardware translator unit rather than the execution engine preferably determines the address of the next intermediate language instructions to translate and provide to the dedicated execution engine. Alternatively the execution engine can determine the next address for the intermediate language instructions.

[0017] In one embodiment, the hardware translator unit only translates some intermediate language instructions, other intermediate language instructions cause a callback to the processor chip that runs a virtual machine to handle these exceptional instructions.

Brief Description of the Drawings

[0018] Fig. 1 is a diagram illustrating a system of one embodiment of the present invention.

[0019] Fig. 2 is a diagram illustrating an Accelerator Chip of one embodiment of the present invention.

[0020] Fig. 3 is a diagram of another embodiment of a system of the present invention.

[0021] Fig. 4A is a state machine diagram illustrating the modes of an Accelerator Chip of one embodiment of the present invention.

[0022] Fig. 4B is a state machine diagram illustrating modes of an accelerator chip of another embodiment of the present invention.

[0023] Fig. 5 is a table illustrating a power management scheme of one embodiment of an Accelerator Chip of the present invention.

[0024] Fig. 6 is a table illustrating one example of a list of bytecodes executed by an Accelerator Chip and a list of bytecodes that cause the callbacks to the processor chip for one embodiment of the system of the present invention.

[0025] Fig. 7 is a diagram that illustrates a common system memory organization for the memory units that can be used with one embodiment of the system of the present invention.

[0026] Fig. 8 is a table of pin functions for one embodiment of an Accelerator Chip of the present invention.

[0027] Fig. 9 is a diagram that illustrates memory wait states for different access times through the accelerator chip or without the accelerator chip for one embodiment of the present invention.

memory units 24. Typically, a processor chip 26 interfaces with memory units 24. This is especially common in embedded systems used for communications, cell phones, personal digital assistants, and the like. In one embodiment the processor chip is a system on a chip (SOC) including a large variety of elements. For example, in one embodiment the processor chip 26 includes a direct memory access unit (DMA) 26a, a central processing unit (CPU) 26b, a digital signal processor unit (DSP) 26c and local memory 26d. In one embodiment, the SOC is a baseband processor for cellular phones for a wireless standard such as GSM, CDMA, W CDMA, GPRS, etc.

[0044] As will be described below, the Accelerator Chip 22 is preferably placed within the path between the processor chip 26 and memory units 24. The Accelerator Chip 22 runs at least portions of programs, such as Java, in an accelerated manner to improve the speed and reduce the power consumption of the entire system. In this embodiment, the Accelerator Chip 22 includes an execution unit 32 to execute intermediate language instructions, and a memory interface unit 30. The memory interface unit 30 allows the execution unit 32 on the Accelerator Chip 22 to access the intermediate language instructions and data to run the programs. Memory interface 30 also allows the processor chip 26 to obtain instructions and data from the memory units 24. The memory interface 30 allows the Accelerator Chip to be easily integrated with existing chip sets (SOC's). The accelerator function can be integrated as a whole or in part on the same chip stack package or on the same silicon with the SOC. Alternatively, it can be integrated into the memory as a chip stack package or on the same silicon.

they are executed on the SoC. Alternatively, the Accelerator Chip of one embodiment can execute every intermediate language instruction.

[0049] Also shown in the execution unit 32 of one embodiment is an interface unit and registers 42. In a preferred embodiment, the processor chip 26 runs a modified virtual machine which is used to give instructions to the Accelerator Chip 22. When a callback occurs, the translator unit 34 sets a register in unit 42 and the execution unit restores all the elements that need restoring and indicates such in the unit 42. In a preferred embodiment, the processor chip 26 has control over the Accelerator Chip 22 through the interface unit and registers 42. The execution unit 32 operates independently once the control is handed over to the Accelerator Chip.

[0050] In a preferred embodiment, an intermediate language instruction cache 38 is used associated with the translator unit 34. Use of an intermediate language instruction cache further speeds up the operation of the system and results in power savings because the intermediate language instructions need not be requested as often from the memory units 24. The intermediate language instructions that are frequently used are kept in the instruction cache 38. In a preferred embodiment, the instruction cache 38 is a two-way associative cache. Also associated with the system is a data cache 40 for storing data.

[0051] Although the translator unit is shown in Fig. 1 as separate from the execution engine, the translator unit can be incorporated into the execution engine. In that case, the central processing unit (CPU) or execution engine has a hardware translator subunit to translate intermediate language instructions into the native instructions operated on by the main portion of the CPU or the execution engine.

[0052] The intermediate language instructions are preferably Java bytecodes. Note that other intermediate language instructions, such as Multos bytecodes, MSIL, BREW, etc., can be used as well. For simplicity, the remainder of the specification describes an embodiment in which Java is used, but other intermediate language instructions can be used as well.

[0053] Figure 2 is a diagram of one embodiment of an Accelerator Chip. In this embodiment, the Java bytecodes are stored in the instruction cache 52. These bytecodes are then sent to the Java translator 34'. A bytecode buffer alignment unit 50 aligns the bytecodes and provides them to the bytecode decode unit 52. In a preferred embodiment,

[0056] When the virtual machine modifies the bytecode to the quick form, the cache line in the hardware accelerator holding the bytecode being modified needs to be invalidated. The same is true when the virtual machine reverses this process and restores the bytecode to the original form. Additionally, the callbacks invalidate the appropriate cache line in the instruction cache using a cache invalidate register in the interface register.

[0057] In some embodiments, when quick bytecodes are used, the modified instructions are stored back into the instruction cache 52. When quick bytecodes are used, the system must keep track of how the Java bytecodes are modified and eventually have instruction consistency between the cache and the external memory.

[0058] In one embodiment, the decoded bytecodes from the bytecode decode unit are sent to a state machine unit and Arithmetic Logic Unit (ALU) in the instruction composition unit 54. The ALU is provided to rearrange the bytecode instructions to make them easier to be operated on by the state machine and perform various arithmetic functions including computing memory references. The state machine converts the bytecodes into native instructions using the lookup table. Thus, the state machine provides an address which indicates the location of the desired native instruction in the microcode look-up table. Counters are maintained to keep a count of how many entries have been placed on the operand stack, as well as to keep track of and update the top of the operand stack in memory and in the register file. In a preferred embodiment, the output of the microcode look-up table is augmented with indications of the registers to be operated on in the register file. The register indications are from the counters and interpreted from bytecodes. To accomplish this, it is necessary to have a hardware indication of which operands and variables are in which entries in the register file. Native Instructions are composed on this basis. Alternately, these register indications can be sent directly to the register file.

[0059] In another embodiment of the present invention, the Stack and Variable manager assigns Stack and Variable values to different registers in the register file. An advantage of this alternate embodiment is that in some cases the Stack and Var values may switch due to an Invoke Call and such a switch can be more efficiently done in the Stack and Var manager rather than producing a number of native instructions to implement this.

[0060] In one embodiment, a number of important values can be stored in the hardware accelerator to aid in the operation of the system. These values stored in the hardware

[0066] The hardware translator unit also preferably stores an indication of the operands and variables stored in the register file of the execution engine. These indications allow the hardware accelerator to compose the converted register-based or native instructions from the incoming stack-based instructions.

[0067] The hardware translator unit also preferably stores an indication of the variable base and operand base in the memory. This allows for the composing of instructions to load and store variables and operands between the register file of the execution engine and the memory. For example, when a variable (Var) is not available in the register file, the hardware issues load instructions. The hardware is adapted to multiply the Var number by four and adding the Var base to produce the memory location of the Var. The instruction produced is based on knowledge that the Var base is in a temporary native execution engine register. The Var number times four can be made available as the immediate field of the native instruction being composed, which may be a memory access instruction with the address being the content of the temporary register holding a pointer to the Vars base plus an immediate offset. Alternatively, the final memory location of the Var may be read by the execution engine as an instruction and then the Var can be loaded.

[0068] In one embodiment, the hardware translator unit marks the variables as modified when updated by the execution of Java bytecodes. The hardware accelerator can copy variables marked as modified to the system memory for some bytecodes.

[0069] In one embodiment, the hardware translator unit composes native instructions wherein the native instruction's operands contain at least two native execution engine register file references where the register file contents are the data for the operand stack and variables.

[0070] In one embodiment a stack-and-variable-register manager maintains indications of what is stored in the variable and stack registers of the register file of the execution engine. This information is then provided to the decode stage and microcode stage in order to help in the decoding of the Java bytecode and generating appropriate native instructions.

[0071] In a preferred embodiment, one of the functions of a Stack-and-Var register manager is to maintain an indication of the top of the stack. Thus, if for example registers R1-R4 store the top 4 stack values from memory or by executing bytecodes, the top of the stack will change as data is loaded into and out of the register file. Thus, register R2 can be

resident in the caches are executed faster and at reduced power consumption because system memory transactions are avoided. Bytecode streams are buffered and analyzed prior to being interpreted using an optimizer based on instruction level parallelism (ILP). The ILP optimizer coupled with locally cached Java data results in the fastest execution possible for each cycle.

[0076] Since the Accelerator Chip is a separate stand-alone Java bytecode execution engine, it processes concurrently while the host microprocessor is either waiting in its polling loop or processing interrupts. Furthermore, the Accelerator Chip is only halted during instances when the host microprocessor needs to access system memory behind it, and the accelerator chip also wants to access system memory at the same time. For example, if the host microprocessor is executing an interrupt service routine or other software from within its own cache, then the Accelerator Chip can concurrently execute bytecodes. Similarly, if Java bytecode instructions and data reside within the Accelerator Chip's internal caches, then the accelerator can concurrently execute bytecodes even if the host microprocessor needs to access system memory behind it.

[0077] Fig. 4A is a state machine showing the two primary modes of the accelerator chip of one embodiment: sleep and running (executing Java bytecode instructions). The accelerator chip automatically transitions between its running and sleep states. In its sleep state, the accelerator chip draws minimal power because the Java engine core and associated components are idled.

[0078] Fig. 4B is a diagram of the states of the accelerator chip of another embodiment of the system of the present invention, further including a standby mode. The standby mode is used during callbacks. In order to reduce power, only the clocks to the Java registers are on. In the standby mode, the processor chip is running the virtual machine to handle the Java bytecode that causes the callback. Since the accelerator chip is in the standby mode, it can quickly recover without having to reset all of the Java registers.

[0079] Fig. 5 shows what components are active and idle in each mode of the state machine of Fig. 4A. When the JVM is not running or when the system determines that additional power savings are appropriate, the Accelerator Chip automatically assumes its sleep mode.

[0083] The Accelerator chip does not disturb interrupt or exception processing, nor does it impose any latency. When an interrupt or exception occurs while the Accelerator Chip is processing, the host microprocessor diverts to an appropriate handler routine without affecting accelerator chip. Upon return from the handler, the host microprocessor returns execution to the software kernel and in turn resumes monitoring the Accelerator Chip. Even when the host microprocessor takes over the memory bus, the Accelerator Chip can continue executing Java bytecodes from its internal cache, which can continue so long as a system memory bus conflict does not arise. If a conflict arises, a stall signal can be asserted to halt the accelerator.

[0084] The Accelerator Chip has several shared registers that are located in its memory map at a fixed offset from a programmable base. The registers control its operation and are not meant for general use, but rather are handled by code within the Software Kernel.

[0085] Referring to Fig. 3, it can be seen that the Accelerator Chip is positioned between the host microprocessor (or the SOC that includes an embedded microprocessor) and the system SRAM and/or Flash memory. All system memory accesses by the host microprocessor therefore pass through the Accelerator Chip. In one embodiment, while fully transparent to all system software, a latency of approximately 4 nanoseconds is introduced for each direction, contributing to a total latency of approximately 8 nanoseconds for each system memory transaction.

[0086] Fig. 6 is a table that illustrates one embodiment of a list of Java bytecodes that are executed by the Java execution unit on the Accelerator Chip and a list of bytecodes that cause a callback to the modified JVM running on the processor chip. Note that the most common bytecodes are executed on the Accelerator Chip. Other less common and more complex bytecodes are executed in software on the processor chip. By excluding certain Java bytecodes from the Accelerator Chip, the Accelerator Chip complexity and power consumption can be reduced.

[0087] Fig. 7 illustrates a typical memory organization and the types of software and data that can be stored in each type of memory. Placement of the items listed in the table below allows the accelerator chip to access the bytecodes and corresponding data items necessary for it to execute Java bytecode instructions.

[0092] Fig. 8 is a table that illustrates on example of the accelerator pin functions for one example of an Accelerator Chip of the present invention.

[0093] In a preferred embodiment, the pins going to the processor chip and going to the memory are located near each other in order to keep the delay through the chip at the minimum for the bypass mode.

[0094] Fig. 9 is a diagram that illustrates the wait states for different access times and bus speeds with an embodiment of a hardware accelerator positioned in between the processor chip, such as an SOC, and the memory. Note that in some cases, additional wait states for access times need to be added due to the introduction of the hardware accelerator in the path between the processor chip and the memory.

[0095] Fig. 10 is a diagram of a hardware accelerator of one embodiment of the present invention. The hardware accelerator 100 includes bypass logic 102. This connects to the system on a chip interface 104 and memory interface 106. The memory controller 108 is interconnected to the interface register 110 which is used to send messages between the system on the chip and the hardware accelerator. Instructions going through the memory controller 108 to the instruction cache 112 and the data from the data cache 114 are sent to the memory controller 108. The intermediate language instructions from an instruction cache 112 are sent to the hardware translator 114, which translates them to native instructions, and sends the translated instructions to the execution engine 116. In this embodiment, the execution engine 116 is broken down into a register read stage 116A, an execution stage 116B and a data cache stage 116C.

[0096] Fig. 11 is a diagram of a hardware accelerator 120 which is used to interface with SRAM memories. Since SRAM memories and SDRAM memories can be significantly different, in one embodiment, there is a dedicated hardware accelerator for each type of memory. Fig. 11 shows the hardware accelerator including an instruction cache, hardware translator, data cache, execution engine, a phase lock loop (PLL) circuit which is used to set the internal clock of the hardware accelerator such that it is synched to an external clock, the interface registers and SRAM slave interface and SRAM master interface. The SRAM slave interface interconnecting to the system on a chip, and SRAM master interface interconnecting to the memory. The diagram of Fig. 11 emphasizes the fact that the connections between the system on a chip and the memory are separate and dealt with

implement them. One new type of library includes graphics for LCD display. For example, a canvas application is used for writing applications that need to handle low-level events and issue graphical calls for drawing on the LCD display. Such an application would typically be used for games and the like. In the embodiment of Fig. 14, a graphics accelerator engine 152 and LCD control and display buffer engines 154 are placed in the hardware accelerator 150, so the control of the system need not be passed to the processor chip. Whenever a graphics element is to be run, a Java program rather than the conventional program is used. The Java program stored in the memory is used to update the LCD display 156. In one embodiment, the Java program uses a special identifier bytecode which is used by the hardware accelerator 150 to determine that the program is for LCD graphics acceleration engine 152. It is not always necessary to have the LCD controller on the same chip if the function is available on the SOC. In this case, only the graphics would still be on the accelerator. The graphics can be for 2D as well as 3D graphics. Additionally, a video camera interface can also be included on the chip. The camera interface unit would interface to a video unit where the video image size can be scaled and/or color space conversion can be applied. By setting certain registers within the accelerator chip it is possible to merge video and graphics to provide certain blend and window effects on the display. The graphics unit would have its own frame buffer and optionally a Z-buffer for 3D. For efficiency, it would be optimal to have the graphics frame buffer in the accelerator chip and have the Z-buffer in the system SRAM or system SDRAM.

[00100] Fig. 15 is a diagram of a chip stack package 160 which includes an accelerator chip 162, flash chip 164 and SRAM chip 166. By putting the accelerator chip 162 in a package along with the memory chips 164 and 166, the number of pins that need to be dedicated on the package for interconnecting between the accelerator chip and the memory can be reduced. In the example of Fig. 15, the reduction in the number of pins allows a set of pins to be used for a bus data and addresses to an auxiliary memory location. Positioning the accelerator chip on the same package as the flash memory chip and SRAM chip also reduces the memory access time for the system.

[00107] Fig. 17 illustrates one example of an execution engine implementing some of the details of the system for the new instructions of Fig. 16A. For the indexed loads, the zero checking logic 170 checks to see whether the value of the index stored in a register, such as register H is 0. When the zero check enable is set (meaning that the instruction is one of the four instructions LDXNC, LWXNC, STXNC, or SWXNC), the zero check enable is set high. Note that the other operations for the load can be done concurrently with this operation. The zero checking logic 170 ensures that the pointer to the array is not 0, which would indicate a null value for the array pointer. When the pointer is correctly initialized, the value will not be a 0 and thus, when the value is a 0, an exception is created.

[00108] The adder/subtractor unit 172 produces a result and also produces the N, Z and C bits which are sent to the N, Z and C logic 174. For the bounds checking case, the bounds checking logic 176 checks to see whether the index is inside the size of the array. In the bounds checking, the index value is subtracted from the array size, the index value will be stored in one register, while the array value is stored in another register. If there is a carry, this indicates an exception, and the bounds check logic 176 produces an index out of range exception when the bounds checking is enabled.

[00109] Logical unit 178 includes the new logic 180. This new logic 180 implements the SGTLT0 and SGTLT0U instructions. Logic 180 uses the N and Z carry bits from a previous subtraction or add. As illustrated by Figs. 16A and 16C, the logic 160 produces a 1, 0 or -1 value, which is then sent to the multiplexer (mux) 182. When the SGTLT0 or SGTLTU instructions are used, the value from the logic 180 is selected by the mux 182.

[00110] Fig. 18A illustrates the Java bytecode instruction IALOAD. The top two entries of the stack are an index and an array reference, which are converted to a single value indicated by the index offset into the array. With the conventional instructions as shown in Fig. 18B, the array reference needs to be compared to 0 to see whether a null pointer exception is to be produced. Next, a branch check is done to determine whether the index is outside of array bounds. The index value address is calculated and then loaded. In Fig. 18C, with the new instructions, the LWXNC reference does a zero check for the register containing the array pointer. The bounds check operation makes sure the index is within the array size. Thereafter the add to determine the address and the load is done.

in system memory requiring several costly memory transactions to execute each Java bytecode instruction. As with bytecode fetches, the memory transactions required to manage and interact with a memory based Java stack are costly in terms of performance and increased system power consumption.

[00116] The Accelerator Chip easily interfaces directly to typical memory system designs and is fully transparent to all system software providing its benefits without requiring any porting or new development tools. Although the JVM is preferably modified to drive Java bytecode execution into the accelerator chip, all other system components and software are unaware of its presence. This allows any and all commercial development tools, operating systems and native application software to run as-is without any changes and without requiring any new tools or software. This also preserves the investment in operating system software, resident applications, debuggers, simulators or other development tools. Introduction of a accelerator chip is also transparent to memory accesses between the host microprocessor and the system memory but may introduce wait states. The Accelerator Chip is useful for mobile/wireless handsets, PDAs and other types of Internet Appliances where performance, device size, component cost, power consumption, ease of integration and time to market are critical design considerations.

[00117] In one embodiment, the accelerator chip is integrated as a chip stack with the processor chip. In another embodiment, the accelerator chip is on the same silicon as the memory. Alternatively, the accelerator chip is integrated as a chip stack with the memory. In a further embodiment, the processor chip is a system on a chip. In an alternative embodiment, the system on a chip is adapted for use in cellular phones.

[00118] In one embodiment, the accelerator chip supports execution of two or more intermediate languages, such as Java bytecodes and MSIL for C#/.NET.

[00119] In one embodiment of the present invention, the system comprises at least one memory, a processor chip operably connected to the at least one memory, and an accelerator chip, the accelerator chip operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory, the accelerator chip being adapted to run at least portions of programs in an intermediate

WHAT IS CLAIMED IS:

1. A system comprising:

at least one memory;

a processor chip operably connected to the at least one memory; and

an accelerator chip, the accelerator chip operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory, the accelerator chip being adapted to run at least portions of programs in an intermediate language.

2. The system of Claim 1 wherein the programs in an intermediate language instructions are Java bytecodes.

3. The system of Claim 2 wherein the processor runs a modified Java virtual machine.

4. The system of Claim 1 wherein the intermediate language is in bytecode form.

5. The system of Claim 1 wherein the accelerator chip is positioned on a memory bus.

6. The system of Claim 1 wherein the memory comprises a number of memory units.

7. The system of Claim 6 wherein the memory units include a static random access memory.

8. The system of Claim 6 wherein the memory units include a flash memory.

18. The system of Claim 17 wherein the accelerator chip includes an instruction cache operably connected to store instructions to be executed within the accelerator chip.

19. The system of Claim 1 wherein the accelerator chip includes a hardware translator unit and a dedicated execution unit adapted to execute native instructions provided by the hardware translator unit, the dedicated execution engine only executing instructions provided by the hardware translator unit.

20. The system of Claim 1 wherein the accelerator chip is integrated as a chip stack with the processor chip.

21. The system of Claim 1 wherein the accelerator chip is on the same silicon as the memory.

22. The system of Claim 1 wherein the accelerator chip is integrated as a chip stack with the memory.

23. The system of Claim 1 wherein the processor chip is a system on a chip.

24. The system of Claim 23 wherein the system on a chip is adapted for use in cellular phones.

25. The system of Claim 1 wherein the accelerator chip supports execution of two or more intermediate languages.

26. The system of Claim 25 wherein the intermediate languages are Java bytecodes and MSIL for C#/.NET.

34. The system of Claim 27 wherein the processor runs a modified virtual machine.

35. The system of Claim 34 wherein the accelerator chip does not execute certain intermediate language instructions and a callback occurs when these intermediate language instructions occur, these intermediate language instructions being executed on the modified virtual machine running on the processor chip.

36. The system of Claim 27 wherein the accelerator chip has a sleep mode with low power consumption.

37. The system of Claim 27 wherein the processor chip is a system on a chip.

38. The system of Claim 27 wherein the accelerator chip includes an interface adapted to allow for memory access for the accelerator chip to at least one memory, and to allow for memory access for the processor chip to the at least one memory.

39. The system of Claim 27 wherein the accelerator chip further includes an instruction cache operably connected to the hardware translator unit storing the intermediate language instructions to be converted.

40. The system of Claim 27 wherein the execution engine is a dedicated execution engine only executing instructions provided by the hardware translator unit.

41. An accelerator chip comprising:

a unit adapted to execute intermediate language instructions; and

an interface, the interface adapted to allow for memory access for the accelerator chip to at least one memory and to allow for memory access for a separate processor chip to the at least one memory.

42. The accelerator chip of Claim 41 wherein the intermediate language instructions are Java bytecodes.

49. The accelerator chip of Claim 48 wherein the intermediate language instructions are Java bytecodes.

50. The accelerator chip of Claim 48 wherein the accelerator chip does not execute every intermediate language instruction but some intermediate language instructions cause a callback to the separate processor chip running a modified virtual machine.

51. The accelerator chip of Claim 48 wherein the accelerator chip has a sleep mode with low power consumption.

52. The accelerator chip of Claim 48 wherein the accelerator chip further includes an instruction cache operably connected to the hardware translator unit storing intermediate language instructions to be converted.

53. The accelerator chip of Claim 48 wherein the execution engine is a dedicated execution engine only executing instructions provided by the hardware translator unit.

54. An accelerator chip comprising:

a hardware translator unit adapted to covert intermediate language instructions into native instructions;

an instruction cache operably connected to the hardware translator unit storing intermediate language instructions to be converted;

an execution engine adapted to execute the native instructions provided by the hardware translator unit; and

an interface, the interface adapted to allow for memory access for the accelerator chip to at least one memory and to allow for memory access for a separate processor chip to the at least one memory.

55. The accelerator chip of Claim 47 wherein the intermediate language instructions are Java bytecodes.

63. The accelerator chip of Claim 59 wherein the accelerator chip includes an interface adapted to allow for memory access for the accelerator chip to at least one memory and allow for memory access for a separate processor chip to the at least one memory.

64. The accelerator chip of Claim 59 wherein the accelerator chip further includes an instruction cache operably connected to the hardware translator unit storing intermediate language instructions to be converted.

65. A method of operating an accelerator chip comprising:

in a hardware translator unit, calculating the address of intermediate language instructions to execute;

obtaining the intermediate language instructions from a memory;

in the hardware translator unit, converting the intermediate language instructions to native instructions;

providing the native instructions to an execution engine; and

in the execution engine, executing the native instructions, wherein for at least one intermediate language instruction a callback to a separate processor chip running a virtual machine is done to handle the intermediate language instruction.

66. The method of Claim 65 wherein the intermediate language instructions are Java bytecodes.

74. A system comprising:

a hardware translator unit adapted to covert intermediate language instructions into native instructions; and

an execution engine adapted to execute the native instructions provided by the hardware translator unit, the execution engine including at least one indexed instruction to do an indexed load from or store into an array, the instruction concurrently checking a first register storing an array pointer to see whether it is null.

75. The system of Claim 74 wherein the hardware translator unit and the execution engine are positioned on an accelerator chip.

76. The system of Claim 74 wherein the accelerator chip is positioned between a processor chip and a memory.

77. The system of Claim 74 wherein the intermediate language instructions are Java instructions.

78. The system of Claim 74 wherein the hardware translator unit translates some array loading and array storing instructions so as to use at least one index instruction.

79. A system comprising:

a hardware translator unit adapted to covert intermediate language instructions into native instructions; and

an execution engine adapted to execute the native instructions provided by the hardware translator unit, the execution engine including at least one indexed instruction to do an indexed load from or store into an array, the execution engine having a zero checking unit adapted to check whether a first register storing an array pointer to see whether it is null, the null checking unit of the execution engine working concurrently with portions of the execution engine doing the indexed load from or store into an array.

88. The system of Claim 86 wherein the signed instruction checks the Z and the N bits. If the Z bit is high, a 0 is put in the register. If the Z bit is low and N is low, 1 is put in the register. If the Z bit is low and N is high, a -1 is put in the register.

89. The system of Claim 86 in which an unsigned instruction check is done to check the Z and C bits. If the Z bit is high, a 0 is put in the register. If the Z bit is low, and C is high, 1 is put in the register. If the Z bit is low and the C is low, -1 is put in the register.

90. The system of Claim 86 in which both signed and unsigned checks are done.

91. The system of Claim 86 wherein the hardware translator unit and execution engine are on an accelerator chip.

92. A system comprising:

at least one memory;

a processor chip operably connected to the at least one memory; and

an accelerator chip, the accelerator chip operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory, the accelerator chip being adapted to run at least portions of programs in an intermediate language, the hardware accelerator including a accelerator of a Java processor for the execution of intermediate language instructions.

97. System of Claim 96 wherein the use of the accelerator chip is in a cell phone or mobile handheld device.

98. A system comprising:

at least one memory;

a processor chip operably connected to the at least one memory; and

an intermediate language accelerator chip, operably connected to the at least one memory, memory access of the processor chip to the at least one memory being sent through the accelerator chip, the accelerator chip having direct access to the at least one memory wherein the accelerator is stacked with one or more memory in the same package.

99. System of Claim 98 wherein the use of the accelerator chip is in a cell phone or mobile handheld device.

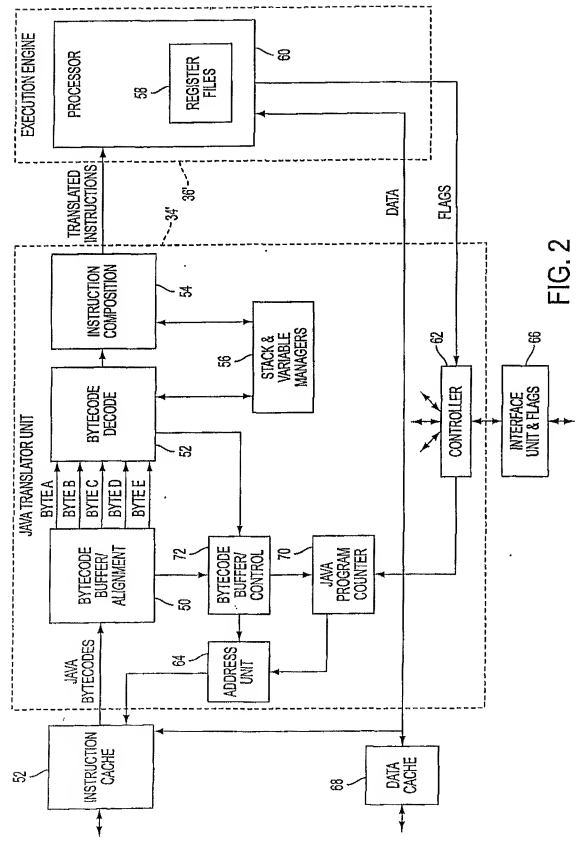
EXECUTION ENGINE
PROCESSOR
58
REGISTER FILES
60
TRANSLATED INSTRUCTIONS
34'
36'
DATA
FLAGS
JAVA TRANSLATOR UNIT
INSTRUCTION COMPOSITION
54
BYTECODE DECODE
52
STACK & VARIABLE MANAGERS
56
BYTE A
BYTE B
BYTE C
BYTE D
BYTE E
BYTECODE BUFFER/ ALIGNMENT
50
72
BYTECODE BUFFER/ CONTROL
70
JAVA PROGRAM COUNTER
JAVA BYTECODES
64
ADDRESS UNIT
62
CONTROLLER
66
INTERFACE UNIT & FLAGS
52
INSTRUCTION CACHE
68
DATA CACHE

FIG. 2

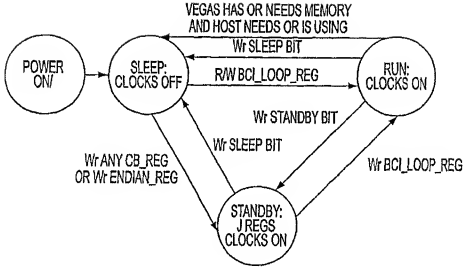
VEGAS HAS OR NEEDS MEMORY
AND HOST NEEDS OR IS USING
Wr SLEEP BIT
POWER
ON
SLEEP:
CLOCKS OFF
R/W BCI_LOOP_REG
RUN:
CLOCKS ON
Wr STANDBY BIT
Wr SLEEP BIT
Wr ANY CB_REG
OR Wr ENDIAN_REG
Wr BCI_LOOP_REG
STANDBY:
J REGS
CLOCKS ON

FIG. 4B

TYPICAL SYSTEM MEMORY ORGANIZATION

SRAM	FLASH
• JAVA HEAP • JAVA STACK • DOWNLOADED JAVA CLASS FILES (INCLUDING THE CONSTANT POOL) • DOWNLOADED JAVA APPLETS/APPLICATIONS	• JAVA RUNTIME ENVIRONMENT (JRE) - JAVA VIRTUAL MACHINE (JVM, CVM,KVM) - JAVA CLASS LIBRARIES - PROFILE CLASS LIBRARIES • DOWNLOADED JAVA CLASS FILES (INCLUDING THE CONSTANT POOL) INTENDED TO BE SAVED AS RESIDENT • DOWNLOADED JAVA APPLETS/APPLICATIONS INTENDED TO BE SAVED AS RESIDENT • OPERATING SYSTEM & DEVICE DRIVERS • NATIVE RESIDENT APPLICATIONS

FIG. 7

JAVA ACCELERATION PIN FUNCTIONS, ORGANIZED BY FUNCTION

PIN	FUNCTION
IO0:IO15	DATA IN/OUT FROM HOST CPU
A0:A23	ADDRESS INPUTS FROM HOST CPU
BHE#, BLE#	BYTE HIGH ENABLE, BYTE LOW ENABLE FROM HOST (ACTIVE LOW)
OE#, WE#	OUTPUT ENABLE, WRITE ENABLE FROM HOST (ACTIVE LOW)
CS0#, CS1#, CS2#, CS3#	CHIP SELECTS 0, 1, 2, 3 FROM HOST (ACTIVE LOW)
Rst#	RESET (ACTIVE LOW)
Clk	CLOCK INPUT
TDI, TDO, TCK, TMS	JTAG SIGNALS: TEST DATA IN, TEST DATA OUT, TEST CLOCK, TEST MODE SELECT
IO0_J:IO15_J	DATA IN/OUT TO MEMORY FROM THE JA108
A0_J:A18_J	ADDRESS OUT TO MEMORY FROM THE JA108
BHE#_J, BLE#_J	BYTE HIGH AND LOW ENABLE TO MEMORY FROM THE JA108
OE#_J, WE#_J	OUTPUT ENABLE, WRITE ENABLE TO MEMORY FROM THE JA108
CS0#_J, CS1#_J, CS2#_J, CS3#_J	CHIP SELECTS 0, 1, 2, 3 TO MEMORY FROM THE JA108
CB	CALLBACK (ACTIVE HIGH) USED TO SIGNAL THAT A BYTECODE NEEDS TO BE EXECUTED BY SOFTWARE RUNNING ON THE HOST MICROPROCESSOR
Vdd,Vss	POWER AND GROUND

FIG. 8

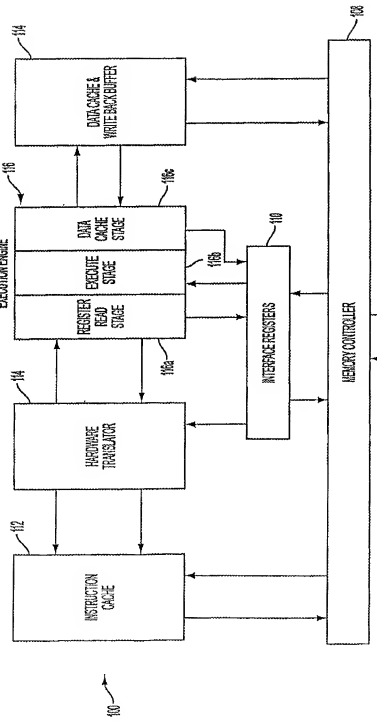
100
112
INSTRUCTION CACHE
114
HARDWARE TRANSLATOR
EXECUTION ENGINE
116
116a
REGISTER READ STAGE
116b
EXECUTE STAGE
116c
DATA CACHE STAGE
114
DATA CACHE & WRITE BACK BUFFER
110
INTERFACE REGISTERS
150
MEMORY CONTROLLER
102
BUS & BYPASS LOGIC
SDRAM/FLASH INTERFACE
160
SOC INTERFACE
104

FIG. 10

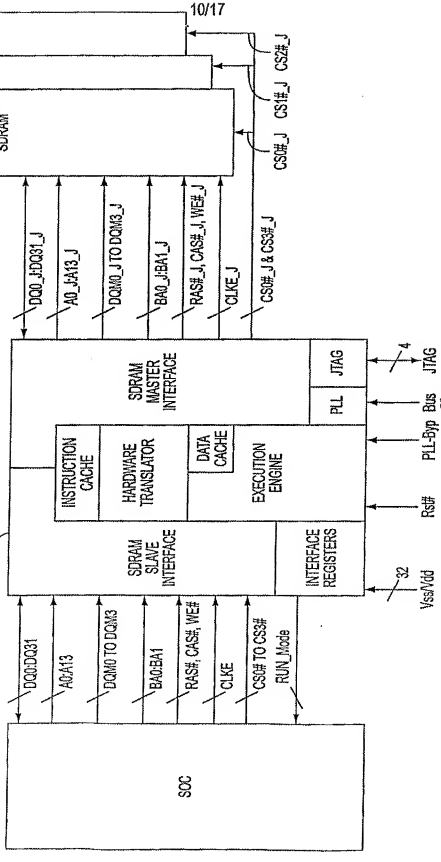
SDFLASH
SDRAM
SDRAM
CS2#_J
CS1#_J
CS#_J
DQ0_J:DQ31_J
A0_J:A13_J
DQM0_J TO DQM3_J
BA0_J:BA1_J
RAS#_J, CAS#_J, WE#_J
CLKE_J
CS0#_J & CS3#_J
SDRAM 32b intf 160 pins
130
INSTRUCTION CACHE
HARDWARE TRANSLATOR
DATA CACHE
EXECUTION ENGINE
SDRAM MASTER INTERFACE
SDRAM SLAVE INTERFACE
INTERFACE REGISTERS
PLL
JTAG
4
JTAG
Bus Clk
PLL-Byp
Rst#
32
Vss/Vdd
DQ0:DQ31
A0:A13
DQM0 TO DQM3
BA0:BA1
RAS#, CAS#, WE#
CLKE
CS0# TO CS3#
RUN_Mode
SOC

FIG. 12

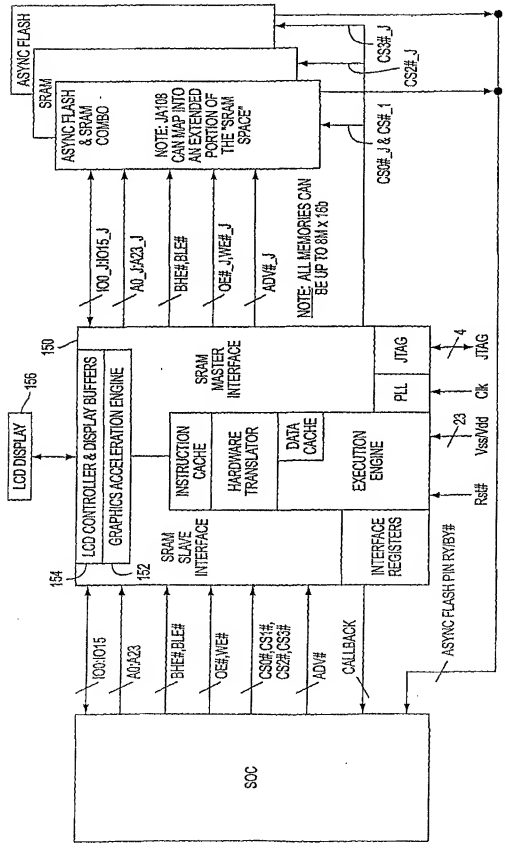
SOC
IO0/IO15
A0/A23
BHE#/BLE#
OE#/WE#
CS0#/CS1#/CS2#/CS3#
ADV#
CALLBACK
ASYNC FLASH RY/BY#
SRAM SLAVE INTERFACE
INTERFACE REGISTERS
INSTRUCTION CACHE
HARDWARE TRANSLATOR
DATA CACHE
EXECUTION ENGINE
SRAM MASTER INTERFACE
PLL
JTAG
LCD CONTROLLER & DISPLAY BUFFERS
GRAPHICS ACCELERATION ENGINE
LCD DISPLAY
150
152
154
156
23
4
Rst#
Vss/Vdd
Clk
JTAG
IO0_IO15_J
A0_A23_J
BHE#_BLE#
OE#_WE#_J
ADV#_J
CS0#_1 & CS#_1
CS3#_J
CS2#_J
NOTE: ALL MEMORIES CAN BE UP TO 8M x 16b
NOTE: JA108 CAN MAP INTO AN EXTENDED PORTION OF THE "SRAM SPACE"
ASYNC FLASH & SRAM COMBO
SRAM
ASYNC FLASH

FIG. 14

SGTLTO	BASED ON C/Z FROM LAST SIGNED ADD/SUB WRITE -1, 0, 1 IN Rd	
SGTLTOU	BASED ON N/Z FROM LAST UNSIGNED ADD/SUB WRITE -1, 0, 1 IN Rd	
LDxNC	LOAD UNSIGNED WORD REGISTER INDEXED	(CHECK REG FOR NULL)
LWxNC	LOAD WORD IMMEDIATE INDEXED	(CHECK REG FOR NULL)
STxNC	STORE WORD REGISTER INDEXED	(CHECK REG FOR NULL)
SWxNC	STORE WORD INDEXED	(CHECK REG FOR NULL)
BNDCK	DO BOUNDS CHECK	

FIG. 16A

SGTLTO

FROM LAST SUB/ADD

N	Z	OUTPUT
x	1	0
0	0	1
1	0	-1

FIG. 16B

SGTLTOU

FROM LAST SUB/ADD

CARRY	Z	OUTPUT
x	1	0
0	0	-1
1	0	1

FIG. 16C

BNDCK

SUBTRACT INDEX FROM ARRAY SIZE

CARRY	
0	NO EXCEPTION
1	EXCEPTION

FIG. 16D

LDxNC
LWxNC
STxNC
SWxNC

ARRAY POINTER	
0	EXCEPTION
NOT ZERO	NO EXCEPTION

FIG. 16E

JAVA BYTECODE INSTRUCTION

IALOAD

FIG. 18A

CONVENTIONAL MICROCODE

```
BEQ     rSRC2, r00, RAISE NULL POINTER EXCEPTION
LWX     rTEMP1, ARRAY_LENGTH_INDEX (rSRC2)
BLT     rSRC1, r00, RAISE A00B EXCEPTION
BGT     rSRC1, rTEMP1, RAISE A00B EXCEPTION
ADDIV   rTEMP1, rSRC2, ARRAY_BASE_BYTE_OFFSET
LDWX    rRes, rSRC1 (rTEMP1)
```

FIG. 18B

MICROCODE WITH NEW INSTRUCTIONS

```
LWXNC   rTEMP1, ARRAY_LENGTH_INDEX (rSRC2)
BNDCK   rTEMP1, rSRC1
ADDIV   rTEMP1, rSRC2, ARRAY_BASE_BYTE_OFFSET
LDWX    rRES, rSRC1 (rTEMP1)
```

FIG. 18C

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/20092

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06 F 9/44
US CL : 717/118
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED
Minimum documentation searched (classification system followed by classification symbols)
U.S. : 717/118, 717/168, 717/108

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
None

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EAST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y,P	US 6,026,485 A (O'CONNOR et al) 15 February 2002, col. 2-5, lines 60-67 and 1-67	1-99
Y,E	US 6,446,192 B1(NARASIMHAM et al.) 03 September 2002, col 2-4, lines 1-67	1-99

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance		
"E"	earlier document published on or after the international filing date	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search 03 SEPTEMBER 2002	Date of mailing of the international search report 18 SEP 2002
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer ANIL KHATRI Telephone No. (703) 305-9282

Form PCT/ISA/210 (second sheet) (July 1998)*